Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Вычисление математических функций с использованием рядов»**

**Выполнил**:

Студент группы 3824Б1ПМ1

Горев А. С.

**Проверила**:

Бусько П.В.

Нижний Новгород

2025

**Содержание**

[Введение 3](#_Toc185261174)

[Постановка задачи 4](#_Toc185261175)

[Руководство пользователя 4](#_Toc185261176)

[Описание программной реализации 7](#_Toc185261177)

[Результаты экспериментов 17](#_Toc185261178)

[Заключение 22](#_Toc185261179)

[Литература 23](#_Toc185261180)

][Приложение 24](#_Toc185261181)

# Введение

В вычислительной математике приближённые методы играют ключевую роль при решении задач, где аналитическое вычисление функций затруднено или невозможно. Одним из таких методов является разложение функций в ряды Тейлора, позволяющее аппроксимировать значения сложных функций с заданной точностью. Данная работа посвящена разработке программы, реализующей вычисление значений функций синуса, косинуса, экспоненты и натурального логарифма через суммирование членов их рядов Тейлора.

Программа демонстрирует, как выбор стратегии суммирования (прямой или обратный порядок) влияет на устойчивость результатов, особенно при работе с ограниченной точностью арифметики с плавающей запятой. Особое внимание уделено обработке граничных условий, таких как область сходимости ряда для натурального логарифма (∣x∣<1) и предотвращению переполнения при вычислении факториалов.

# Постановка задачи

1. Реализовать на языке программирования C вычисление математических функций с использованием рядов Тейлора, включая:
   * Синус sin(*x*)
   * Косинус cos(*x*)
   * Экспоненту ()
   * Натуральный логарифм ln(1+*x*)

C применением двух методов суммирования:

* + Прямое (последовательное сложение членов ряда от первого к последнему)
  + Обратное (сложение членов ряда от последнего к первому).

1. Разработать программу, которая:
   * Запрашивает у пользователя значение аргумента x и количество итераций.
   * Вычисляет приближенные значения функций с использованием обоих методов суммирования.
   * Сравнивает результаты с точными значениями (через библиотечные функции math.h), рассчитывая погрешность для каждого метода.

# Руководство пользователя

**Описание работы программы.**

Программа предназначена для вычисления значений функций с помощью ряда Тейлора. Кроме того, программа вычисляет погрешность на каждом шаге итерации, сравнивая результат с точным значением функции, полученным с помощью стандартных математических функций.

**Действия пользователя:**

**Шаг 1**: Запуск программы

Запустите программу. В консоли появится меню с предложением выбрать функцию для вычисления или выйти из программы:

1. Синус

2. Косинус

3. Экспонента

4. Натуральный логарифм.

5. Выход

**Шаг 2**: Выбор функции и ввод параметров

1. Введите номер действия (от 1 до 5) и нажмите Enter.

Если выбран натуральный логарифм (4), программа проверит условие |x| < 1. При нарушении условия выдаст ошибку и вернёт в меню.

2. Введите значение x (например, 0.5 для логарифма или 3.14 для синуса) и нажмите Enter.

3. Введите количество итераций count (натуральное число). Программа проверит корректность ввода.

**Шаг 3**: Выполнение вычислений

1. Программа вычислит ряд Тейлора для выбранной функции двумя способами:

Прямое суммирование — члены ряда добавляются последовательно, начиная с первого.

Обратное суммирование — члены ряда добавляются в обратном порядке (с последнего к первому).

2. На экран будут выводиться значения в формате:

<Сумма на i-й итерации> <Ошибка>

**Шаг 4**: Просмотр результатов

1. После завершения вычислений программа выведет результаты для прямого и обратного суммирования, а также величину ошибки.

2. Программа автоматически вернётся в меню выбора функции для новых вычислений.

**Шаг 5**: Завершение работы

1. Для выхода из программы введите номер 5.

2. Программа завершит работу с сообщением: Выполнен выход из программы.

# Описание программной реализации

**1. Файл «Lab\_2.h»**

В нем находятся заголовочные файлы (*рис.1*), структура (*Рис. 2)* и прототипы функций (*рис.3*).

Изображение выглядит как Шрифт, текст, рукописный текст, снимок экрана

Автоматически созданное описание

*Рис. 1 Заголовочные файлы*

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

*Рис. 2 Структура*

*Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание*

*Рис. 3 Прототипы функций*

Подключенные библиотеки:

* <stdio.h>, используется для ввода и вывода данных в консольном приложении;
* <stdlib.h>, используется для работы с динамической памятью;
* <math.h>, предоставляет набор математических функций и констант, которые используются для выполнения различных математических операций;

Описание структуры:

В программе используется структура Xn, которая предназначена для хранения данных, связанных с вычислением значений функций с использованием рядов Тейлора.

Поля структуры:

* int count: Хранит количество итераций (количество членов ряда, используемых для вычисления).
* double x: Хранит значение аргумента x, для которого вычисляется функция.
* double\* err: Указатель на массив, в котором хранятся значения погрешностей на каждом шаге итерации.
* double\* sum: Указатель на массив, в котором хранятся промежуточные суммы ряда на каждом шаге итерации.

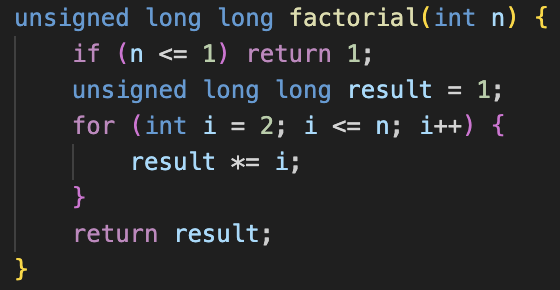
**2. Файл «Lab\_2\_Realiz»**

* Функция «factorial» (*рис.4*)

В качестве значения аргумента принимает:

int n: числовое значение до которого вычисляется факториал

Функция вычисляет факториал числа.



*Рис.4 Функция «factorial»*

* Функция «sinXn»(*рис.5*)

void sinXn(int count, double x, double\* err, double\* sum);

В качестве значений аргумента принимает:

* int count: количество итераций суммирования ряда Тейлора (определяет точность вычислений).
* double x: аргумент функции синус, для которого вычисляется приближённое значение.
* double\* err: указатель на массив для записи погрешностей на каждой итерации (разница между текущей суммой и sin(x)).
* double\* sum: указатель на массив для записи частичных сумм ряда после добавления каждого члена.

Функция вычисляет частичные суммы ряда Тейлора для sin(x) прямым порядком (от первого члена к последнему).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.5 Функция «sinXn»*

* Функция «sinXnRev»(*рис.6*)

void sinXnRev(int count, double x, double\* err, double\* sum);

В качестве значений аргумента принимает:

* int count: количество итераций суммирования ряда Тейлора (определяет точность вычислений).
* double x: аргумент функции синус, для которого вычисляется приближённое значение.
* double\* err: указатель на массив для записи погрешностей на каждой итерации (разница между текущей суммой и sin(x)).
* double\* sum: указатель на массив для записи частичных сумм ряда после добавления каждого члена.

Функция вычисляет частичные суммы ряда Тейлора для sin(x) в обратном порядке (от последнего члена к первому).

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

*Рис.6 Функция «sinXnRev»*

* Функция «cosXn»(*рис.7*)

void cosXn(int count, double x, double\* err, double\* sum);

В качестве значений аргумента принимает:

* int count: количество итераций суммирования ряда Тейлора (определяет точность вычислений).
* double x: аргумент функции косинус, для которого вычисляется приближённое значение.
* double\* err: указатель на массив для записи погрешностей на каждой итерации (разница между текущей суммой и cos(x)).
* double\* sum: указатель на массив для записи частичных сумм ряда после добавления каждого члена.

Функция вычисляет частичные суммы ряда Тейлора для cos(x) прямым порядком (от первого члена к последнему).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.7 Функция «cosXn»*

* Функция «cosXnRev»(*рис.8*)

void cosXnRev(int count, double x, double\* err, double\* sum);

В качестве значений аргумента принимает:

* int count: количество итераций суммирования ряда Тейлора (определяет точность вычислений).
* double x: аргумент функции косинус, для которого вычисляется приближённое значение.
* double\* err: указатель на массив для записи погрешностей на каждой итерации (разница между текущей суммой и cos(x)).
* double\* sum: указатель на массив для записи частичных сумм ряда после добавления каждого члена.

Функция вычисляет частичные суммы ряда Тейлора для cos(x) в обратном порядке (от последнего члена к первому).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.8 Функция «cosXnRev»*

* Функция «expXn»(*рис.9*)

void expXn(int count, double x, double\* err, double\* sum);

В качестве значений аргумента принимает:

* int count: количество итераций суммирования ряда Тейлора (определяет точность вычислений).
* double x: аргумент функции экспоненты, для которого вычисляется приближённое значение.
* double\* err: указатель на массив для записи погрешностей на каждой итерации (разница между текущей суммой и exp(x)).
* double\* sum: указатель на массив для записи частичных сумм ряда после добавления каждого члена.

Функция вычисляет частичные суммы ряда Тейлора для exp(x) прямым порядком (от первого члена к последнему).

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

*Рис.9 Функция «expXn»*

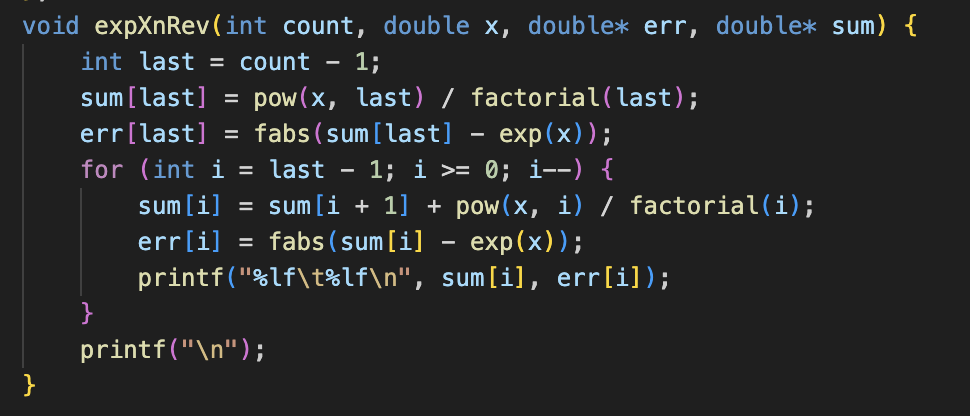
* Функция «expXnRev»(*рис.10*)

void expXnRev(int count, double x, double\* err, double\* sum);

В качестве значений аргумента принимает:

* int count: количество итераций суммирования ряда Тейлора (определяет точность вычислений).
* double x: аргумент функции экспоненты, для которого вычисляется приближённое значение.
* double\* err: указатель на массив для записи погрешностей на каждой итерации (разница между текущей суммой и exp(x)).
* double\* sum: указатель на массив для записи частичных сумм ряда после добавления каждого члена.

Функция вычисляет частичные суммы ряда Тейлора для exp(x) в обратном порядке (от последнего члена к первому).



*Рис.10 Функция «expXnRev»*

* Функция «lnXn»(*рис.11*)

void lnXn(int count, double x, double\* err, double\* sum);

В качестве значений аргумента принимает:

* int count: количество итераций суммирования ряда Тейлора (определяет точность вычислений).
* double x: аргумент функции натурального логарифма, для которого вычисляется приближённое значение.
* double\* err: указатель на массив для записи погрешностей на каждой итерации (разница между текущей суммой и ln(x+1)).
* double\* sum: указатель на массив для записи частичных сумм ряда после добавления каждого члена.

Функция вычисляет частичные суммы ряда Тейлора для ln(x+1) прямым порядком (от первого члена к последнему).

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

*Рис.11 Функция «lnXn»*

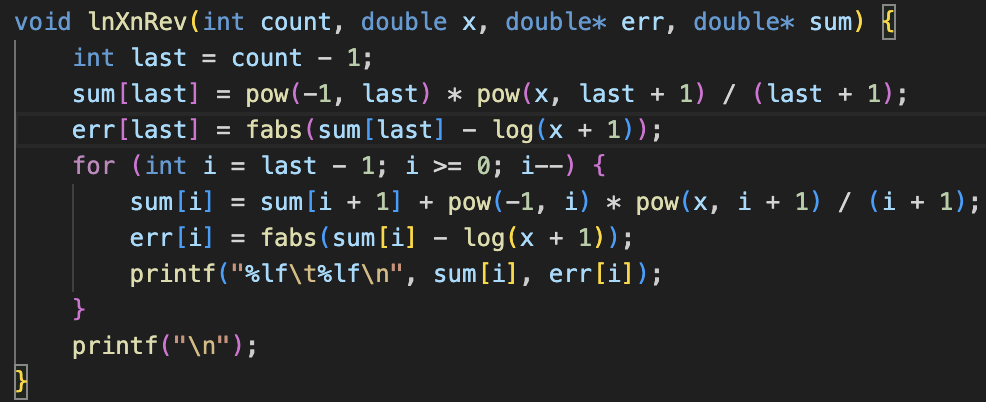
* Функция «lnXnRev»(*рис.12*)

void lnXnRev(int count, double x, double\* err, double\* sum);

В качестве значений аргумента принимает:

* int count: количество итераций суммирования ряда Тейлора (определяет точность вычислений).
* double x: аргумент функции натурального логарифма, для которого вычисляется приближённое значение.
* double\* err: указатель на массив для записи погрешностей на каждой итерации (разница между текущей суммой и ln(x+1)).
* double\* sum: указатель на массив для записи частичных сумм ряда после добавления каждого члена.

Функция вычисляет частичные суммы ряда Тейлора для ln(x+1) в обратном порядке (от последнего члена к первому).



*Рис.12 Функция «lnXnRev»*

* Функция «create\_array»(*рис.13*)

void create\_array(struct Xn\* array, int count);

В качестве значений аргумента принимает:

* struct Xn\* array: указатель на структуру Xn, для которой выделяется память.
* int count: количество элементов в массивах sum (частичные суммы) и err (погрешности).

Функция выделяет динамическую память для массивов sum и err внутри структуры Xn, инициализирует элементы массивов нулевыми значениями и устанавливает поле count структуры равным переданному значению count.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.13 Функция «create\_array»*

* Функция «free\_array»(*рис.14*)

void free\_array(struct Xn\* array);

В качестве значений аргумента принимает:

* struct Xn\* array: указатель на структуру Xn, память которой требуется освободить.

Функция освобождает динамическую память, выделенную для массивов sum и err внутри структуры Xn, обнуляет указатели на массивы (sum = NULL, err = NULL) и сбрасывает значение поля count структуры до нуля.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.14 Функция «free\_array»*

1. **Файл «Lab\_2\_main.cpp»**

Файл «Lab\_2\_main.cpp». Он включает заголовочный файл «Lab\_2.h» и содержит функцию main, реализующую консольное приложение для работы с рядами Тейлора. Принцип работы:

1**.** Выбор функции для вычисления:

Пользователю предлагается выбрать одну из четырёх функций или выйти из программы:

* Синус.
* Косинус.
* Экспонента.
* Натуральный логарифм.
* Выход из программы.

Ввод считывается в переменную choice.

2**.** Ввод параметров:

Для выбранной функции пользователь вводит значение x. Для натурального логарифма проверяется условие |x| < 1. При нарушении выводятся ошибка и повторный запрос. Вводится количество итераций count (натуральное число).

3. Инициализация массивов:

Вызывается функция create\_array, которая выделяет память для массивов sum (частичные суммы) и err (погрешности) внутри структуры Xn.

4. Вычисление рядов Тейлора:

Для выбранной функции вызываются два метода суммирования:

1. Прямой порядок (например, sinXn).
2. Обратный порядок (например, sinXnRev).

На каждой итерации:

* Рассчитывается текущая сумма ряда.
* Вычисляется погрешность как разница между суммой и точным значением функции.

5. Вывод результатов:

Результаты выводятся в виде:

<Сумма на итерации> <Погрешность>

Пример:

0.141120 0.000005

0.141119 0.000004

6. Освобождение памяти:

После завершения вычислений вызывается free\_array, которая освобождает память, выделенную для массивов sum и err.

7. Завершение работы:

Для выхода из программы пользователь вводит «5». Программа завершает работу с сообщением: «Выполнен выход из программы».

# Результаты экспериментов

Для проведения эксперимента рассмотрим погрешность между нашей и библиотечной реализациями значений функций в конкретной точке. Для функций возьмем . Я рассмотрел все предложенные способы суммирования с разным числом слагаемых.

Все данные представлены в таблицах и на графиках. Ось абсцисс отвечает за число слагаемых, а ось ординат – за погрешность.

**1. Прямое суммирование.**

1. Функция sin(x) (*таблица 1, график 1*).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 1.641120 |
| 2 | 0.383880 |
| 3 | 0.050049 |
| 4 | 0.004192 |
| 5 | 0.000245 |
| 6 | 0.000011 |
| 7 | 0.000000 |
| 8 | 0.000000 |
| 9 | 0.000000 |
| 10 | 0.000000 |

*График 1*

*Таблица 1*

На *Графике 1* представлен график зависимости погрешности от числа слагаемых для функции sin(x). Видно, что уже при 6 слагаемых точность достаточно высока и практически не отличается от истинной.

1. Функция cos(x) (*таблица 2, график 2*).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 2,510008 |
| 2 | 0,864992 |
| 3 | 0,147508 |
| 4 | 0,015216 |
| 5 | 0,001057 |
| 6 | 0,000053 |
| 7 | 0,000002 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |

*График 2*

*Таблица 2*

Аналогично с синусом, ряд с косинусом очень быстро сходится (см. График 2).

1. Функция (*таблица 3, график 3*).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 16,085537 |
| 2 | 11,585537 |
| 3 | 7,085537 |
| 4 | 3,710537 |
| 5 | 1,685537 |
| 6 | 0,673037 |
| 7 | 0,239108 |
| 8 | 0,076385 |
| 9 | 0,022144 |
| 10 | 0,005872 |

*График 3*

*Таблица 3*

С экспонентой ситуация несколько иная. На *Графике 3* видно, что кривая дольше прижимается к оси абсцисс, по сравнению с двумя предыдущими функциями, которые мы уже рассмотрели.

1. Функция (*таблица 4, график 4*).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,030465 |
| 2 | 0,011202 |
| 3 | 0,004423 |
| 4 | 0,001827 |
| 5 | 0,000778 |
| 6 | 0,000338 |
| 7 | 0,00015 |
| 8 | 0,000067 |
| 9 | 0,00003 |
| 10 | 0,000014 |

*График 4*

*Таблица 4*

График для логарифма точно также плавно опускается (см. *График 4*). Точность очень высокая уже с третьего слагаемого.

1. **Обратное суммирование**

a. Функция sin(x) (*таблица 5, график 5*).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,14112 |
| 2 | 0,14112 |
| 3 | 0,141131 |
| 4 | 0,140875 |
| 5 | 0,145312 |
| 6 | 0,091071 |
| 7 | 0,525 |
| 8 | 1,5 |
| 9 | 3 |
| 10 | 0 |

*График 5*

*Таблица 5*

На *Графике 5* видно, что при 10 слагаемых значение ряда абсолютно не отличается от истинного значения.

b. Функция cos(x) (*таблица 6, график 6*).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,989992 |
| 2 | 0,989994 |
| 3 | 0,98994 |
| 4 | 0,991049 |
| 5 | 0,974777 |
| 6 | 1,1375 |
| 7 | 0,125 |
| 8 | 3,5 |
| 9 | 1 |
| 10 | 0 |

*График 6*

*Таблица 6*

Точность такая же высокая, ряд с 7-го слагаемого близок к истине, а начиная с 10 точность абсолютная (см. *График 6*).

c. Функция (*таблица 7, график 7*).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 20,015024 |
| 2 | 19,8523 |
| 3 | 19,418372 |
| 4 | 18,405872 |
| 5 | 16,380872 |
| 6 | 13,005872 |
| 7 | 8,505872 |
| 8 | 4,005872 |
| 9 | 1,005872 |
| 10 | 0,005872 |

*График 7*

*Таблица 7*

Можем наблюдать на графике, что с увеличением количества слагаемых точность также кратно растет.

d. Функция (*таблица 8, график 8*).

|  |  |
| --- | --- |
| Число слагаемых | Погрешность |
| 1 | 0,405518 |
| 2 | 0,405301 |
| 3 | 0,40579 |
| 4 | 0,404674 |
| 5 | 0,407278 |
| 6 | 0,401028 |
| 7 | 0,416653 |
| 8 | 0,374986 |
| 9 | 0,499986 |
| 10 | 0,000014 |

*График 8*

*Таблица 8*

Погрешность колеблется, но уже к 10-му слагаемому значение очень близко к истинному.

# Заключение

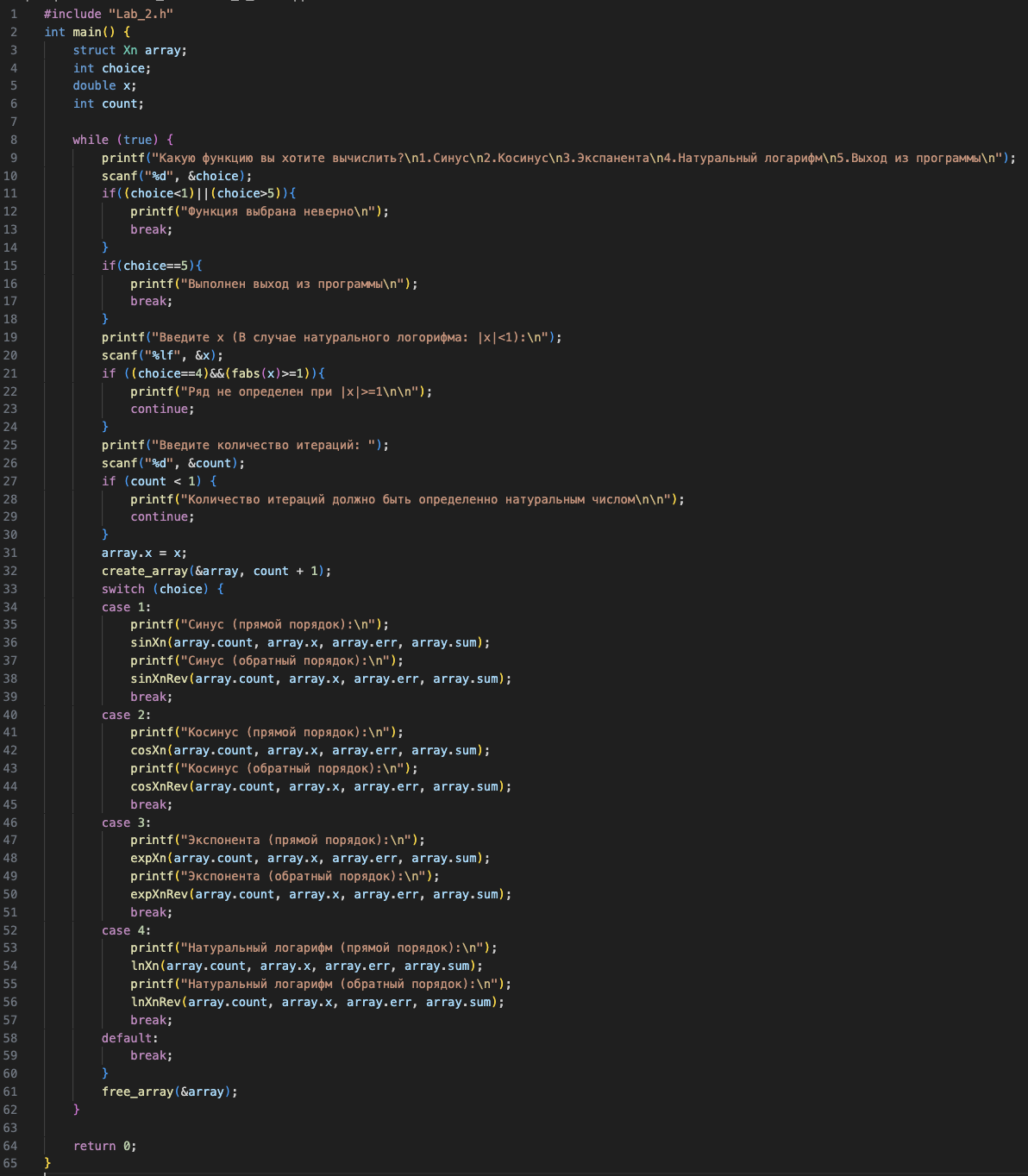
В рамках лабораторной работы мы реализовали вычисление математических функций с использованием рядов Тейлора и провели сравнение результатов, полученных разными методами суммирования, и библиотечной функции из «math.h».

Анализируя два метода суммирования, я пришёл к выводу, что их результаты очень близки к друг другу, а различия между ними минимальны. Основным фактором, влияющим на точность вычислений, является количество слагаемых в ряде: чем больше слагаемых используется, тем точнее результат.

# Литература

1. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования СИ //М.: Финансы и статистика. – 1992.
2. Кнут Д. Э. Искусство программирования: Сортировка и поиск. – Издательский дом Вильямс, 2000. – Т. 3.
3. Дж. Клейнберг, Э. Тардос. Алгоритмы: разработка и применение.
4. Р. Мартин. Чистый код.

# Приложение



*Рис. 15 Функция main*